A Handheld Computer as an Interaction Device to a Virtual Environment

Kent Watsen, Rudolph P. Darken, and Michael V. Capps
Department of Computer Science
Naval Postgraduate School
Monterey, California 93943-5118
+1 831 656 4072
watsen | darken | capps@cs.nps.navy.mil

Abstract

A fundamental problem hindering the advancement of virtual world development is that of interaction techniques. There is contention between 2D and 3D techniques and uncertainty as to which is appropriate and when. We have developed a simple mechanism to address this problem whereby the user performs tasks appropriate to 2D interfaces with the 3Com PalmPilot handheld computer. The use of a wireless serial connection allows for unencumbered immersion in CAVE-like environments. Our implementation utilizes Bamboo, a dynamically extensible virtual environment toolkit, which enables our design to accommodate new user interfaces on the fly. We are in the early stages of analyzing these tasks and techniques for usability and efficiency. The paper reports techniques that we have implemented, and the specifics of using Bamboo and a PalmPilot for virtual world applications.

CR Categories: H.5.2 [Information Interfaces and Presentation]: User Interfaces - GUI; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - Virtual reality; C.5.3 [Computer System Implementation]: Microcomputers - Portable Devices.

Additional Keywords: Virtual Environments, Interaction Techniques, Mobile Computing, Dynamic Extensibility, Component System

1 Introduction

One of the fundamental problems that continues to plague the development of virtual environments (VEs) is that of interaction techniques. There is contention between two-dimensional (2D) and three-dimensional (3D) techniques and extreme uncertainty as to which is appropriate and when. Consequently, there are numerous instances of 3D devices being used for 2D techniques such as gloves and trackers used for selecting items from virtual menus. While it is certainly the spatial attributes of VEs that makes them special, not all interaction tasks are necessarily 3D. Many are inherently 2D and thus are best executed with 2D techniques preferably using 2D devices. However, this seems to cause a conflict in that 3D devices are needed for the 3D components of the interface. This paper discusses methods for accommodating 2D interaction techniques at the same time.

1.1 Interaction in Virtual Environments

The types of tasks that users might perform in any interactive system are independent of the dimensionality of the system. Specifically, Foley *et al.* [FW84] identified interaction tasks as:

- **Selection**: Make a selection from a number of alternatives.
- **Position**: Indicate a position on the display or in the workspace.
- **Orientation**: Alter the orientation of an object in the workspace.
- **Path**: Generate a path, which is a series of positions and orientations over time.
- **Quantification**: Specify a value (i.e. a number) to quantify a measure.
- **Text**: Input a text string.

While this enumeration predates the proliferation of 3D interfaces, it remains complete and accurate for today's interactive systems. When applied to VEs, some of these items are clearly 3D in nature (e.g. position, orientation, and path), while others are not (e.g. selection, quantification, and text). It may be convenient to use traditional 2D techniques (e.g. menus and dialog boxes) for the 2D elements of a 3D interface, but as these techniques were designed for use with 2D devices (e.g. mouse or trackball), using a 3D device (e.g. flying mouse or instrumented glove) may prove unsuitable.

There is clearly a conflict in the design of interaction techniques to accomplish these tasks in that techniques that work well for 2D tasks can often be cumbersome, or at least somewhat unnatural, when applied to a 3D environment using 3D devices. The converse is equally true. The development of 3D techniques using 2D devices has been the subject of significant previous research. Given the constraint that a 2D device must be used for a 3D task, these techniques are excellent, but if this constraint could be removed, we can do better.

When taking the approach of replicating 2D interaction techniques in a 3D environment, the user is forced to sacrifice performance on one type of task in lieu of another. It may be easy to manipulate a virtual object, for instance, but difficult to pick an item from a menu. In some cases, this may be acceptable if it has been determined that either the 2D or 3D components of the interface are predominant such that the performance sacrifice is minimal or non-existent. But this is rarely the case. Most interfaces have both 2D and 3D elements that the user must be able to execute to a high degree of proficiency. However, it has often been assumed impractical to require the simultaneous use of both conventional 2D (e.g. mouse, keyboard) and 3D (e.g. tracker, glove) devices for VE applications. We would like to reach a compromise that maps 2D tasks to 2D devices and 3D tasks to 3D devices in a practical way.

2 Previous Work

Designing interaction techniques for VEs in this manner—that is, retaining 2D characteristics for 2D tasks without losing 3D capabilities for 3D tasks—is not an entirely new idea.

An early attempt was to implement traditional 2D widgets in 3D, but to replace the pointing/selection device with a spatial tracker or glove [JE92]. This work was later improved upon by using Motif widgets instead of true 3D panels, and using Fakespace Pinch™ gloves for selection [CR96]. The motivation for these projects incorporated two essential assumptions:

1. 2D GUI widgets are a familiar interaction technique for most users, and
2. there is a need to keep interaction uniform, such that 3D objects are manipulated similarly to 2D widgets.

Though the first assumption seems correct at first glance, it does not always hold true. In this case, while "Virtual Motif" may resemble the conventional Motif interface, it doesn't behave similarly. Interaction is more than the look—it's also the feel. These techniques are not only cumbersome to use but they steal precious screen space away from the content of the VE. It can also be argued that the second assumption is simply a false constraint resulting from a technological artifact. We assume that 2D and 3D have to be the same because we assume the use of a single interaction device. There is no reason to believe that the use of a 2D device precludes the use of a 3D device in every case. The real world has both 2D and 3D objects and interactions (e.g. writing on a pad of paper versus picking up a pencil) -- so should the virtual world.

There have been several attempts to overcome problems associated with using 2D widgets in 3D spaces. One design addressed only menus by using a 2D textual overlay on the 3D world activated by a simple speech recognition system [DR94]. This is a useful but incomplete solution. The more well known method, and the one most closely associated with our work, is the "Virtual Tricorder" which confines 2D interaction and information to a virtual handheld object [WG95, AS95] (see Figure 1A). This technique is very effective but is hampered by the low resolution of head-mounted displays. Fine detail is commonly lost on objects such as maps or text. However, the Virtual Tricorder has other advantages such as being able to integrate 3D functionality such as that shown in 3D magic lenses [VJ96] (see Figure 1B). Here, the position of the handheld device in virtual space is used as part of an interaction technique. This same technique has been used in the real world in an augmented reality application using a position tracked handheld computer as a window on the world [RN95].

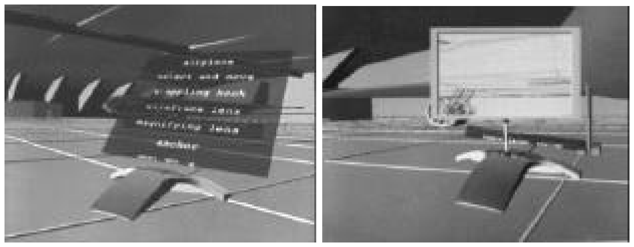

Figure 1: A. The virtual tricorder used for menu selection. B. The virtual tricorder used with a wireframe magic lens [WG95].

The technique we have developed is a combination of all these that allows the 2D components of the interface to be performed in a purely 2D fashion, and the 3D components to be performed in 3D.

3 Design

The method we have developed involves the use of a 3Com PalmPilot handheld computer as an interaction device to the VE. This approach allows us to use conventional 2D widgets such as menus and dialog boxes that maintain not only the look but also the feel of the original technique. In fact, 2D techniques for our application are identical to 2D techniques used for any Pilot application. We also have the added advantage that we do not lose screen space for the 2D part of the interface. The display remains in the user's hand at all times. As this approach requires an unobstructed view of the handheld, it is not compatible with some display devices such as opaque HMDs (head mounted displays). It has, however, been shown to be compatible with desktop, CAVE-like [CS93], and AR (augmented reality) display environments.

Implementing an interface on a PDA (personal digital assistant), such as 3Com's PalmPilot, is conceptually similar to implementing an interface using a standard GUI toolkit such as X-Windows or MS-Windows. Following is a brief discussion of how typical GUIs are developed, so as to facilitate understanding of the subtleties in our approach.

3.1 Traditional GUIs

In a traditional GUI, user input is translated by the UI into an event (e.g. button pressed), which is then passed down to the underlying application. The application translates the abstract event into an action (e.g. select object) and passes state information back up to the UI for display. This scenario is illustrated in Figure 2 below.

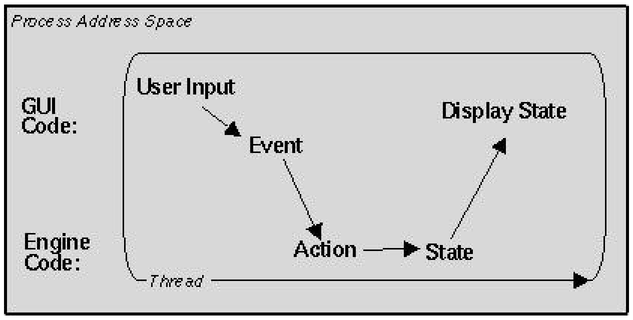

Figure 2: One process, one thread

For small projects, the event handling code may be hard coded directly into the underlying application. Doing so, though, not only leads to a highly coupled system that is more difficult to maintain, but it also ties the refresh rate of both to the slower of the two. For this reason, larger projects typically break the system into application-specific and application-neutral parts - the UI and the underlying "engine" respectively. In order to allow the UI and engine to refresh as quickly as possible, each is given its own execution loop. Traditionally, this meant that each would have its own heavyweight process and would communicate with the other using some form of inter-process communication (IPC) such as shared memory. In order to reduce interdependencies, a client stub in the engine's process space translates the UI events into engine-specific commands and returns the new state information for display. This approach is depicted in Figure 3 below.

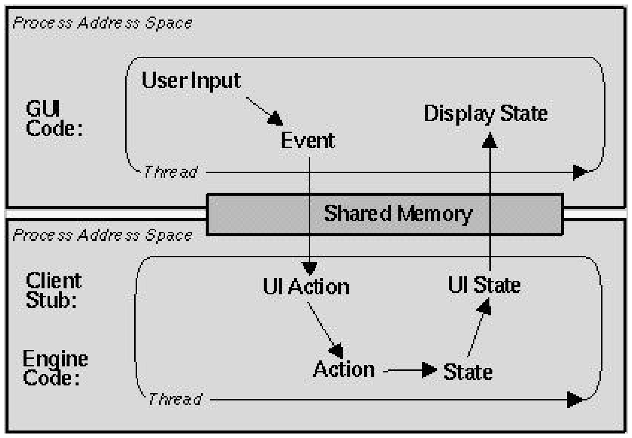

Figure 3: Two processes, one threads each

However, the recent availability of lightweight threads enables multiple control loops to exist within the same process space. More specifically, the UI and the engine control loops can be threads in the same process. Being in the same process space removes the need for a client stub and thus represents the best approach. For completeness, this approach is depicted in Figure 4 below.

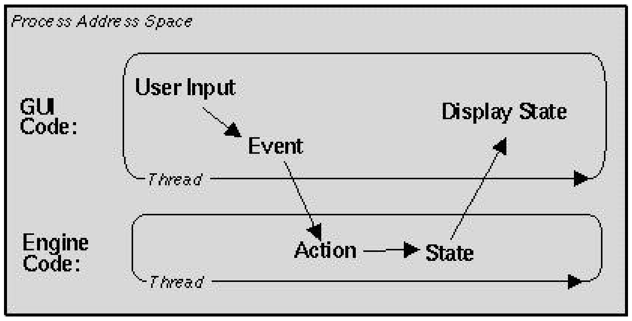

Figure 4: One process, two threads

3.2 Incorporating the PalmPilot

The PalmPilot computer handles events generated both by user input and applications. As such, it is very much like a GUI running in its own process address space. Thus, our design must model after the 2 process approach above (see Fig. 3). A significant difference exists, however, in that the PDA must communicate with its host computer using a serial port. Fortunately, we can easily accommodate this by having the client stub be in its own control loop so that it can actively poll the serial port. This scenario is illustrated in Figure 5 below.

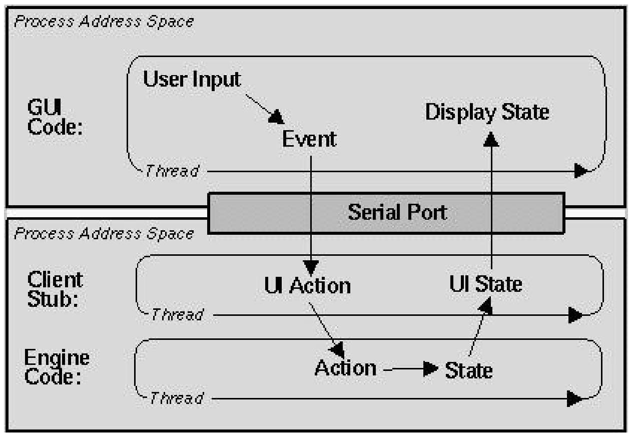

Figure 5: Two Processes, three threads

3.3 Extending Our Requirements for the GUI

The approach developed in the previous section is viable, but it is not satisfying as it neither completely decouples the system nor considers the long-term future of VEs. The first failing is best exemplified by the fact that the client stub must be compiled into the engine's process space and therefore limits the range of interactions until it is replaced. The second failing is grounded in the observation that sooner or later there will exist large-scale, global, virtual environments that, because they are simultaneously inhabited by millions, can never shutdown or reboot for any reason. In such an environment, all modifications (e.g. system configurations) would have to occur "on the fly." Specifically, our client stub must be dynamically extensible; capable of supporting new GUIs that may be developed long after its initial deployment.

4 Implementation

The last section extended the scope for the implementation of user interfaces for VEs. This section introduces Bamboo and shows how it is used to implement fully decoupled and dynamically extensible interfaces for the PalmPilot.

4.1 Bamboo

Bamboo [WZ98] is a multi-platform system supporting real-time, networked, virtual environments. Unlike previous efforts, this design focuses on the ability for the system to dynamically configure itself without explicit user interaction, allowing applications to take on new functionality after execution. In particular, this framework facilitates the discovery of virtual environments on the network at runtime.

Fundamentally, Bamboo moves dynamically loadable libraries into and out of memory. Each library is part of a "module," which is a directory structure that contains, in addition to the library itself, the geometry, texture, sound, and/or anything else used by it. Typically, modules are archived and subsequently downloaded via HTTP over the Internet.

In general, modules depend on one or more other modules much like an application depends on one or more libraries. As the dependent module(s) must be in memory before the new module is loaded, Bamboo defines a convention whereby each module need only specify its immediate dependencies (name, version, URL), as oppose to all the dependencies of all the modules it depends on. These dependencies are loaded first, even if it is necessary to download them from the Internet.

Finally, as it is often not enough to simply bring executable code into the process's memory, Bamboo defines an executable lattice that the newly loaded code can attach itself to. Alternatively, the newly loaded code could spawn its own lightweight thread. Also, being concerned with concurrency issues surrounding threads, Bamboo implements all the necessary synchronization mechanisms.

4.2 Implementing a UI for a PalmPilot

The initial concern was that the client stub was hard-coded into the engine and therefore may not be usable with other GUIs. What is needed is a convention whereby client stubs can plug into the system. Fortunately, on a PalmPilot each "applet" can be designed to implement a specific interaction capability (e.g. a selection operation). If each applet has a one-to-one relationship to some client proxy, then we need only load the proxy when the applet is loaded and unload it when the applet is unloaded.

From the system's perspective, a special client stub is loaded to implement a simple message passing convention. In fact, there are exactly three messages that this client stub might have to interpret:

1. `loadModule`
   - load specified module, download as necessary
2. `unloadModule`
   - unload specified module
3. `passEventToModule`
   - pass numBytes to specified module

When the user selects an applet to run, the applet sends a `loadModule` message through the serial port to the special client stub, which returns a handle to the module if successfully loaded and an error code otherwise. When the user closes an applet, the applet sends an `unloadModule` message, specifying itself using the handle it received when loaded. At all other times, the user can only be using one applet at a time. Every time the user interacts with that applet, it generates and sends a `passEventToModule` message to its specific client proxy module, which presumably will know how to deal with that message and send back an appropriate response.

However, this solution only solves our first concern. The second concern was that the GUI itself would have to be dynamically extensible. For instance, after selecting an object in a VE, it would be nice if an object specific UI could be sent to the Pilot for further user interaction. Using our current approach, this is easily accomplished by sending a new applet to the Pilot. The next time the user looks in their applets folder, they could choose to load the new applet. Implementing this is as easy having each applet also support the standard “hot-synch” function provided with the system.

4.3 Current Software Implementation

Our current implementation consists of three applets and the default LaunchPad applet built into the pilot (see Figure 6).

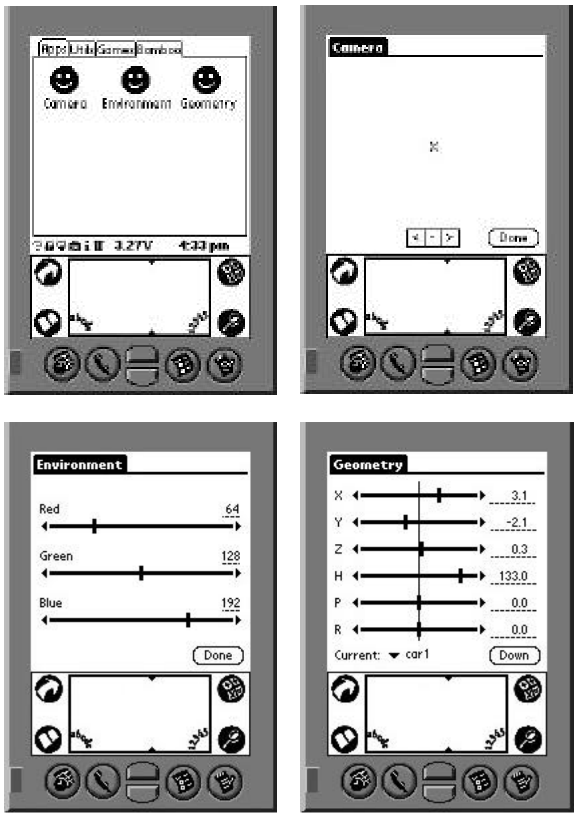

Figure 6: the PalmPilot’s LaunchPad and the Camera, Environment, and Geometry applets.

The user begins by selecting which of the applets to run from the LaunchPad, which will become visible again when the user selects the “Done” pushbutton (i.e. Bamboo applet is closed) or when the built-in “Applications” silk-screen button is selected. New applets pushed to the PalmPilot will be visible for selection the next time the user runs the LaunchPad applet.

The Camera applet, when loaded, initializes itself by asking its client stub to find and attach itself to the camera. The applet and its client also initialize themselves to be in the “no move” state, which is indicated to the user by the middle toggle button near the bottom of the screen being selected. At this point, the user can move the stylus around the screen, which has the effect of panning the camera around the world (i.e. moving up the screen reflects pitching up in the VE). At any time, the user can toggle between the “move backward” (<), “no move” (-), and the “move forward” (>) states while panning around the screen. In this way, the user can “fly” around the VE.

The Environment applet, when loaded, initializes itself by asking its client stub to discover and report the current clear color, which is then applied to the three sliderbars (red, green, blue) before the user allowed to interact with the applet (note that this is unnoticeably fast). As depicted by the screen capture (Fig. 6), each sliderbar has an associated field that displays its value (0-255). The user can also directly input a value into the field, which then updates the sliderbar. In this way, the user can change the background clear color to be any of 16 million colors.

The Geometry applet, when loaded, initializes itself by asking its client stub to discover and report the number of objects in the VE. An object from this list can then be selected from the object list at the bottom of the screen, which itself is initialized to “<none>.” Once an object is selected, the client stub reports it current location (X, Y, and Z) and orientation (H, P, and R). These values (floats) are applied to each sliderbar and its associated field located to its right. Again, the user can move either the sliderbar or directly input the value in the field. In this way, any object in the VE can be placed anywhere in the VE.

The three applets were developed using Metrowerk’s CodeWarrior for PalmPilot integrated development environment. Each of these applets were developed in parallel with their client stub. The process of creating a new applet/stub combination is simplified, as Bamboo encapsulates the common functionality and establishes conventions for client stub interoperation. This enables the developer to focus on their particular application task.

4.4 Current Hardware Implementation

As Bamboo is available on multiple platforms, the machine's vendor and operating system are not important; we were able to run on both an SGI and a Windows NT machines. The machine must have a free serial port capable of running at least at 9600 baud. The Pilot can connect to the serial port using the standard Pilot cable. However, as our goal is to have the user to be able to move freely about the physical environment (e.g. a CAVE), we also spliced a cable to work with a wireless serial port. We selected ABACOM Technologies' Rtcom2-RS232, which has the benefit of transparently working just like a standard serial port (i.e. no software modifications were required). For the display, we back-projected a InFocus LitePro 730 against a Stewert FilmScreen 170 SN.T in a dark room. This is illustrated in Figure Y.

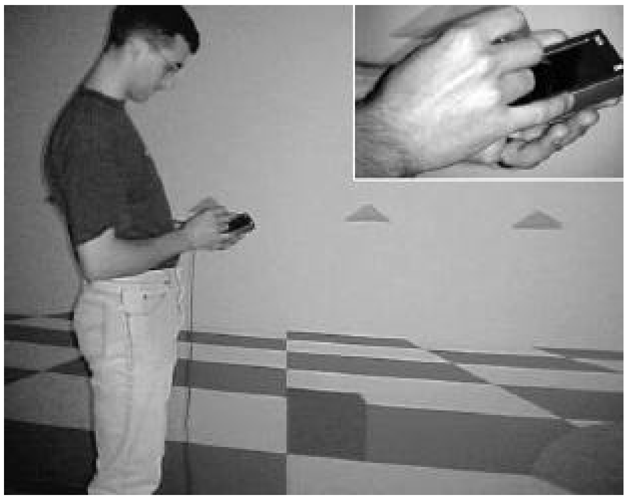

Figure 7: User interacting with an immersive VE via PDA

5 Conclusions

Using a PDA as an interface in a VE enables traditional UI techniques—without forcing 2D metaphor in a 3D space nor 3D metaphor in a 2D space. Applying good software engineering practices allowed us to decouple the application dependent and independent parts. Finally, recognizing the need for dynamic extensibility, our solution takes on an implementation that is fully reconfigurable. Although we have just begun and have yet to formalize a user study, this approach has already shown great promise. It should probably be mentioned that Bamboo's Java-based GUI is also fully dynamically-extensible and has been shown to make certain kinds of intra-user interactions never before realized possible.

There are, however, several limitations to this approach. Because an unobstructed view of the PDA is required, this system is not compatible with some display devices, such as HMDs. Also, because our projection display system has limited luminance, our PDA had to be back-lit. Furthermore, the wireless serial port we used carries a fair amount of noise that causes quite a few packets (messages) to be dropped. Finally, our software running on the PalmPilot currently does not support more then one applet to be open at a time or enable new applets to be pushed to it at runtime.

Finally, as we have just started to integrate tracker technologies, we have yet to resolve what hybrid interaction metaphors do or do not make sense. We are currently pursuing letting the user "fly" by selecting a velocity and simply pointing in the direction to move. We are also pursuing letting the user directly manipulate objects via the PDA (i.e. rotating the PDA rotates the selected object), like when using a 3D mouse. Obviously it will take a long time for the community to decide which of these metaphors are useful. We can at least state use of an inertial tracker eliminates the interference problems associated with traditional magnetic trackers.

5.1 Future Work

There are a few issues that we have addressed but not completely resolved and a few issues that we have not addressed at all. A glaring area for exploration is that area surrounding the integration of a tracker. Almost as importantly, we'd like to further observe the ability to push applets to the PDA of the fly, especially in the context of providing object or user specific interactions. Our current camera interface should be extended to allow the user to select which of multiple cameras is to be manipulated. Similarly, our environment module should enable the user to affect other "environment" specific controls, such as fog and time of day. Some interesting new areas for research include the use of a color PDA (PalmPilot is rumored to release one next year) and the IR port, although we doubt the user will be gain anything after already having the wireless serial port.

5.2 Availability

Bamboo is an active project at the Naval Postgraduate School and is currently the infrastructure for several large projects in the United States and abroad. Primary interest in Bamboo is attributed to the simplicity of its design, availability on multiple platforms, and never before seen flexibility as users can leverage each other's efforts from anywhere on the Internet. The standard distribution is a single source tree with multiple targets. Although the code is primarily C++, the GUI is written in Java and other components use Tcl, Python, and Perl, There is no licensing fee or shareware charge. A mailing list (with monthly web archives) has been established so that interested developers can freely exchange comments. Plans are being made to provide ongoing support and maintenance; developments will be announced on the mailing list as they become known. Additional papers, documents, and the modules identified in this paper may be found at http://npsnet.nps.navy.mil/Bamboo.

Acknowledgements

Special recognition is given to Todd Gagnon for the wireless serial port and tracker integration. As always, the patience of Dr. Mike Zyda has been appreciated. Finally, this effort could not have been without the generous support of our sponsors: DARPA, ONR, DMSO, Advanced Network & Services, and the National Tele-Immersion Initiative.

References

[AS95] Angus, I.G. and H.A. Sowizral, “Embedding the 2D Interaction Metaphor in a Real 3D Virtual Environment.” *Proceedings of SPIE*, 1995. 2409: pp. 282-293.

[CR96] Coninx, K., F. van Reeth, and E. Flerackers, “2D Human-Computer Interaction Techniques in Immersive Virtual Environments.” *Proceedings of Computer Graphics 96*, 1996.

[CS93] Cruz-Neira, C., D. Sandin, and T. DeFanti, “Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE.” *Computer Graphics*, 1993: p. 135-142.

[DR94] Darken, R.P. “Hands-Off Interaction with Menus in Virtual Spaces.” *Proceedings of SPIE 1994, Stereoscopic Displays and Virtual Reality Systems*, 1994. 2177: p. 365-371.

[FW84] Foley, J.D., V.L. Wallace, and P. Chan, “The Human Factors of Computer Graphics Interaction Techniques,” *IEEE Computer Graphics & Applications*, 1984. pp. 13-47.

[JE92] Jacoby, R.H. and S.R. Ellis, “Using Virtual Menus in a Virtual Environment.” *Proceedings of SPIE Technical Conference 1666*, 1992.

[RN95] Rekimoto, J. and K. Nagao, “The World Through the Computer: Computer Augmented Interaction with Real World Environments.” *Proceedings of UIST 95*, 1995: p. 29-36.

[VJ96] Viega, J., et al., “3D Magic Lenses.” *Proceedings of UIST 96*, 1996: p. 51-57.

[WG95] Wloka, M.M. and E. Greenfield, “The Virtual Tricorder: A Uniform Interface for Virtual Reality.” *Proceedings of UIST 95*, 1995: p. 39-40.

[WZ98] Watsen, K. and M. Zyda. “Bamboo - A Portable System for Dynamically Extensible, Real-time, Networked, Virtual Environments.” *IEEE Virtual Reality Annual International Symposium (VRAIS'98)*, Atlanta, Georgia, March 1998.